On the Parameterised Complexity of String Morphism Problems

Henning Fernau[1], Markus L. Schmid[1], and Yngve Villanger[2]

1 **Fachbereich IV – Abteilung Informatikwissenschaften**
Universität Trier, Trier, Germany
{Fernau,MSchmid}@uni-trier.de
2 **Department of Informatics**
University of Bergen, Bergen, Norway
yngve.villanger@gmail.com

Abstract

Given a source string u and a target string w, to decide whether w can be obtained by applying a string morphism on u (i. e., uniformly replacing the symbols in u by strings) constitutes an NP-complete problem. For example, the target string $w :=$ `baaba` can be obtained from the source string $u :=$ `aba`, by replacing `a` and `b` in u by the strings `ba` and `a`, respectively. In this paper, we contribute to the recently started investigation of the computational complexity of the string morphism problem by studying it in the framework of parameterised complexity.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes, F.2.2 Nonnumerical Algorithms and Problems, F.4.3 Formal Languages

Keywords and phrases String Problems, String Morphisms, Parameterised Complexity, Exponential Time Hypothesis, Pattern Languages

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.55

1 Introduction

Many of the typical string problems are concerned with special kinds of string operations like, e. g., concatenating strings with each other, deleting symbols from or inserting symbols into a string or replacing symbols by other symbols or even by other strings. Among the most prominent of these string problems are *string-to-string correction*, *sequence alignment* as well as the *longest common subsequence* and *shortest common supersequence problem*. The complexity of these problems have been intensely studied, both in the classical sense as well as in the parameterised setting (see, e. g., [1, 9]).

In this work, we investigate string problems that arise from a less well-known operation on strings, i. e., mapping a *source string* u to a *target string* w by uniformly (i. e., by a mapping) replacing the symbols of u by strings. For example, we can turn the source string $u :=$ `abba` into the target string $w :=$ `bbaaaaabba` by replacing `a` and `b` of u by the strings `bba` and `aa`, respectively. On the other hand, $w' :=$ `abaaaaaabb` cannot be obtained from u in a similar way. The *string morphism problem* (denoted by StrMorph) is to decide for two given strings u and w, whether or not w can be obtained from u by this kind of operation. Due to its simple definition, variants of this $\mathcal{NP}$-complete problem can be found in many different areas of theoretical computer science. In fact, many respective results are scattered throughout the literature without pointers to each other and consulting the existing literature suggests that variants of the string morphism problem have emerged and have been investigated in different contexts without knowledge of other related work.

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 55–66

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A Brief History of the String Morphism Problem

The origin of the string morphism problem is usually traced back to the 1979 paper by Angluin [3], in which she introduced the model of *pattern languages* (the membership problem of which is essentially the string morphism problem), but, independently and at the same time, it has also been studied by Ehrenfeucht and Rozenberg in [11]. Garey and Johnson [17], by referring to a private communication with Aho and Ullman from 1977, report the $\mathcal{NP}$-completeness of the problem Regular Expression Substitution, for which, on close inspection, the string morphism problem turns out to be a natural subproblem.

Since their introduction, Angluin's pattern languages have been intensely studied in the context of learning theory and formal language theory. While questions of learnability as well as language theoretical properties were the main focus of research, results regarding the complexity of their membership problem (except of its general $\mathcal{NP}$-completeness) were sparse and only appeared as by-products (see, e. g., [3,18,20,28]). In the pattern matching community, independent of Angluin's work, the string morphism problem has been investigated in terms of a special kind of pattern matching paradigm, called parameterised pattern matching (see [2, 4, 8, 13]). The string morphism problem can also be seen as the solvability problem for word equations where one side does not contain variables (for more details on word equations, see Mateescu and Salomaa [24]). Combinatorial properties of the operation of uniformly replacing the symbols in a string by other strings are investigated in numerous other areas of theoretical computer science and discrete mathematics, such as (un-)avoidable patterns (cf. Jiang et al. [22]), the ambiguity of morphisms (cf. Freydenberger et al. [16]) and equality sets (cf. Harju and Karhumäki [19]). Last but not least, the string morphism problem can also be found in practical applications. More precisely, it constitutes a special case of the matchtest for *regular expressions with backreferences* (see, e. g., Câmpeanu et al. [6]), which nowadays are a standard element of most text editors and programming languages.

The Contribution of this Paper

A systematic study of the computational complexity of StrMorph has been taken on just recently. In [25–27], several possibilities are presented of how to restrict the structure of the source strings, such that StrMorph can be solved in polynomial time, and in [13], the $\mathcal{NP}$-completeness of a large number of strongly restricted versions of StrMorph is shown.

In the literature mentioned above, different *variants* of StrMorph are considered, each tailored to different aspects and research questions. The most common variants arise from whether or not we allow symbols to be *erased* (i. e., replaced by the empty string), whether or not we allow *constants* (also called *terminals*) in the source string, which cannot be replaced and whether or not the replacement function needs to be *injective* (i. e., different symbols cannot be replaced by the same string). While the subtle difference of whether or not symbols can also be erased has a substantial impact on decidability questions for pattern languages, the differentiation of StrMorph in an injective and a non-injective version is motivated by pattern matching tasks. In addition to these different variants of StrMorph, we can observe many natural *parameters* (in the definition of the following parameters, let u be a source string over the alphabet A and let w be a target string over the alphabet B).

- p_1: The cardinality of A.
- p_2: The length of w.
- p_3: The cardinality of B.
- p_4: The maximum length of the strings substituted for the symbols in u.
- p_5: The maximum number of occurrences of any symbol in u.

In [13], for every variant of StrMorph and for every subset of the above parameters, it is shown whether the chosen parameters can be bounded by constants such that the resulting version of StrMorph can be solved in polynomial time or whether it is still $\mathcal{NP}$-complete. For example, if the cardinality of A is bounded by a constant, then all variants of StrMorph can be solved in polynomial time. This trivially follows from the fact that there is a simple brute-force algorithm that runs in time that is exponential only in p_1. Close inspection reveals that StrMorph can also be solved in time that is exponential only in p_2 (for details, the reader is referred to [18] and also [13]). Consequently, in terms of parameterised complexity, StrMorph parameterised by p_1 or p_2 is in XP and the question arises whether these problems are in FPT. Unfortunately, as a main result of this work, we report that both of the above parametrisations are $W[1]$-hard, even if additional parameters are taken into account. More precisely, we show that the following versions of StrMorph are $W[1]$-hard:

1. *all variants* of StrMorph parameterised by (p_1, p_3, p_5),
2. *all variants* of StrMorph (except the nonerasing ones) parameterised by (p_2, p_3, p_4, p_5).

For obtaining the first result, we extend a result by Stephan et al. [29], who showed the $W[1]$-hardness for a special variant of StrMorph parameterised by p_1 and in order to prove the second result, we devise a new reduction from k-Multicoloured-Clique. With respect to the parameters defined above, this leaves only very few parameterised variants of StrMorph that could possibly be in FPT. In this regard, we report the fixed parameter tractability of the following versions of StrMorph:

3. *all variants* of StrMorph parameterised by (p_1, p_2),
4. *all variants* of StrMorph parameterised by (p_1, p_4),
5. *all nonerasing variants* of StrMorph parameterised by p_2,
6. *the nonerasing, injective variant* of StrMorph parameterised by (p_3, p_4).

The above results 1 to 6 (in conjunction with results from [13]) completely settle the fixed parameter tractability of all possible parameterised variants of StrMorph, with respect to the parameters p_1 to p_5. We complement these results by showing for all the $W[1]$-hard cases $W[1]$-membership or $W[P]$-membership.

We conclude the paper by demonstrating the unlikeness of a subexponential algorithm for an important variant of the string morphism problem by applying the *Exponential Time Hypothesis*. Due to space constraints, for most of our results we only provide proof sketches.

2 Preliminaries

Let $\mathbb{N} := \{1, 2, 3, \ldots\}$. For an arbitrary alphabet A, a *string* (*over* A) is a finite sequence of symbols from A, and ε is the *empty string*. The notation A^+ refers to the set of all non-empty strings over A, and $A^* := A^+ \cup \{\varepsilon\}$. For the *concatenation* of two strings w_1, w_2 we write $w_1\, w_2$. We say that a string $v \in A^*$ is a *substring* (or *factor*) of a string $w \in A^*$ if there are $u_1, u_2 \in A^*$ such that $w = u_1\, v\, u_2$. The powers w^i, $i \in \mathbb{N}$, of a string w are inductively defined by $w^1 := w$ and $w^i = w\, w^{i-1}$. The notation $|K|$ stands for the size of a set K or the length of a string K. By $|w|_b$, we denote the number of occurrences of $b \in A$ in w, by $w[i, j]$, we denote the factor from position i to j in w and $w[i] := w[i, i]$. Let A and B be alphabets. A mapping $h : A^* \to B^*$ with $h(u\, w) = h(u)\, h(w)$, for every $u, w \in A$, is a *morphism*. It can be easily verified that a morphism is uniquely defined by the images $h(b)$, $b \in A$. If, for every $b \in A$, $h(b) \neq \varepsilon$, then h is said to be *nonerasing*. If, for all $b, c \in A$ with $b \neq c$, $h(b) \neq \varepsilon$ and $h(c) \neq \varepsilon$ implies $h(b) \neq h(c)$, then h is E-*injective* and h is *injective* if it is E-injective and nonerasing. The *size* of a morphism h is defined by $|h| := \max\{|h(b)| \mid b \in A\}$. Let A and B

be alphabets with $B \subseteq A$. A morphism $h : A^* \to B^*$ that satisfies $h(b) = b$, for every $b \in B$, is a *substitution*.

▶ **Example 1.** Let $u_1 := x\,x\,y\,z\,y$, $u_2 := x\,\mathtt{a}\,y\,\mathtt{b}\,y\,x\,y$, $w_1 := \mathtt{abababababab}$, $w_2 := \mathtt{bacbabbacb}$ and $w_3 := \mathtt{abaabbbababab}$. The nonerasing morphism h_1 defined by $h_1(x) := h_1(y) := h_1(z) := \mathtt{ab}$ satisfies $h_1(u_1) = w_1$. However, u_1 cannot be mapped to $w_1\mathtt{b} =: w_1'$ by a nonerasing morphism. If, on the other hand, we do not restrict ourselves to nonerasing morphisms, then $h_2(u_1) = w_1'$, where h_2 is defined by $h_2(x) := h_2(y) := \varepsilon$ and $h_2(z) := w_1'$. It can be verified that $g_1(u_2) = w_2$, where g_1 is a substitution defined by $g_1(x) := \mathtt{bacb}$, $g_1(y) := \varepsilon$ and $g_2(u_2) = w_3$, where g_2 is a substitution defined by $g_2(x) := g_2(y) := \mathtt{ab}$. Furthermore, u_2 cannot be mapped to w_2 by a nonerasing substitution and u_2 cannot be mapped to w_3 by an E-injective or injective substitution.

Next, we define several variants of string morphism problems. The following is the most general string morphism problem, which shall serve as a base for the definitions of all the further restricted versions.

StrMorph
Instance: Two strings u and w over some alphabets A and B.
Question: Does there exist a morphism $h : A^* \to B^*$ with $h(u) = w$?

By StrSubst, we denote the version of StrMorph, where instead for a morphism we are looking for a substitution. By adding the prefixes NE, Inj and Ne-Inj, we denote the variants of the problems StrMorph and StrSubst, where the morphism (the substitution) needs to be nonerasing, E-injective and nonerasing injective, respectively. Let SMP be the class containing exactly these 8 variants of the string morphism problem, i. e.,

$$\text{SMP} := \{Z\text{-StrMorph}, Z\text{-StrSubst} \mid Z \in \{\varepsilon, \text{Ne}, \text{Inj}, \text{Ne-Inj}\}\}.$$

Next, we fix some notation that shall be used throughout the paper. For an instance (u, w) of one of the above defined string morphism problems, u is called the *source string*, w is called the *target string* and the respective alphabets A and B with $u \in A^*$ and $w \in B^*$ are called the *source* and *target alphabet*, respectively. From now on, the target alphabet is always denoted by Σ and the source alphabet is X for string morphism problems and $(X \cup \Sigma)$ for string substitution problems, where $X \subseteq \{x_1, x_2, x_3, \ldots\}$. The symbols in Σ are called *terminals* and the symbols in X are called *variables*. For any string $u \in (\Sigma \cup X)^*$, by $\text{var}(u)$ we refer to the set of variables occurring in u and $|u|_{\text{var}}$ is the maximum number of occurrences of a variable in u, i. e., $|u|_{\text{var}} := \max\{|u|_x \mid x \in \text{var}(u)\}$.

For the problems in SMP, we consider the following parameters: $|\text{var}(u)|$ (the number of variables in the source string), $|\Sigma|$ (the cardinality of the target alphabet), $|w|$ (the length of the target string), $|u|_{\text{var}}$ (the maximum number of occurrences of a variable) and $|h|$ (the size of the morphism or substitution). A *list of parameters* is a tuple $[\rho_1, \rho_2, \ldots, \rho_k]$, where $1 \leq k \leq 5$, $\{\rho_1, \rho_2, \ldots, \rho_k\} \subseteq \{|\text{var}(u)|, |\Sigma|, |w|, |u|_{\text{var}}, |h|\}$. For example, $[|\text{var}(u)|, |\Sigma|, |h|]$ is a list of parameters. For every $K \in \text{SMP}$ and every list L of parameters, by L-K, we denote the problem K parameterised by the parameters in L, e. g., $[|\Sigma|, |u|_{\text{var}}]$-Ne-StrMorph is the parameterised version of Ne-StrMorph, where the cardinality of the target alphabet and the maximum number of occurrences per variable are parameters. We wish to point out that all parameters but $|h|$ are implicitly given by the source and target string. In particular, for negative instances of the string morphism problems, the parameter $|h|$ is undefined. Hence, as a convention, whenever we consider parameterised problems L-K, $K \in \text{SMP}$, where L

contains $|h|$, we assume that the parameter $|h|$ is explicitly given as input along with the source and target string.

In the following, we briefly recall some of the main concepts of parameterised complexity theory (for an overview, the reader is referred to Flum and Grohe [14]). The class of fixed parameter tractable problems is denoted by FPT and in order to show fixed parameter *intractability*, we use the class $W[1]$. The Clique problem parameterised by the size of the clique is denoted by k-Clique and k-Multicoloured-Clique is the problem to decide for a graph $\mathcal{G} := (V, E)$ and a partition $V_1, V_2, \ldots, V_k$ of V, such that every V_i is an independent set, whether or not $\mathcal{G}$ has a clique of size k. It is a well-known fact that both k-Clique and k-Multicoloured-Clique are complete for $W[1]$ (with respect to parameterised reductions) [12]. Another $W[1]$-complete problem that we use is Short-NTM-Comp, i. e., to decide for a nondeterministic Turing machine M, a string w over the input alphabet of M and a parameter $k \in \mathbb{N}$ whether or not M has an accepting computation for w with at most k steps (see Cai et al. [5], Downey et al. [10], Cesati [7]). The classes of polynomial and nondeterministically polynomial time solvable problems are denoted by $\mathcal{P}$ and $\mathcal{NP}$, respectively.

In Section 4, in order to argue for the unlikeness of a subexponential algorithm, we shall apply the *Exponential Time Hypothesis* (ETH) by Impagliazzo, Paturi, and Zane [21], which, informally speaking, is the conjecture that 3SAT cannot be solved in time $2^{o(n)}$. For an introduction to ETH, the reader is referred to [15, 23].

3 The Parameterised Complexity of String Morphism Problems

In this section, we show for *every* list of parameters L and for *every* $K \in$ SMP, whether or not L-K is fixed parameter tractable or $W[1]$-hard. We start with the hardness results and then present fpt-algorithms for all the other cases. This section is concluded by showing $W[1]$-membership and $W[P]$-membership for the $W[1]$-hard cases.

3.1 $W[1]$-Hardness

As explained in Section 1, it can be easily seen that all variants of StrMorph can be solved in polynomial time if $|\mathrm{var}(u)|$ or $|w|$ is bounded by a constant. Furthermore, as shall be explained in Section 3.2, it also follows trivially that all variants of StrMorph are in FPT if parameterised by $|\mathrm{var}(u)|$ *and* $|w|$ at the same time. Hence, the most interesting question is whether this also holds if either $|\mathrm{var}(u)|$ or $|w|$ is a parameter. We shall show that this is very unlikely, since the corresponding parameterised versions of the string morphism problems are $W[1]$-hard. First, we consider the case that $|\mathrm{var}(u)|$ is a parameter and $|w|$ is not a parameter, for which we can show $W[1]$-hardness, even if $|u|_{\mathrm{var}}$ and $|\Sigma|$ are parameters, too.

▶ **Theorem 2.** *For every $K \in$ SMP, $[|\mathrm{var}(u)|, |\Sigma|, |u|_{\mathrm{var}}]$-$K$ is $W[1]$-hard.*

In [29], Stephan et al. give a reduction from k-Clique to $[|\mathrm{var}(u)|, |\Sigma|, |u|_{\mathrm{var}}]$-NE-StrSubst, which proves its $W[1]$-hardness. This reduction can be extended to a parameterised reduction for all problems $[|\mathrm{var}(u)|, |\Sigma|, |u|_{\mathrm{var}}]$-$K$, $K \in$ SMP, which implies Theorem 2.

Next, we consider the case where $|w|$ is a parameter instead of $|\mathrm{var}(u)|$. In this regard, we can state a rather strong result, i. e., the $W[1]$-hardness for all (but the nonerasing) variants of string morphism problems parameterised by all the considered parameters except $|\mathrm{var}(u)|$.

▶ **Theorem 3.** *For every $Z \in \{\mathrm{Inj}, \varepsilon\}$ and $K \in \{\mathrm{StrMorph}, \mathrm{StrSubst}\}$, the problem $[|w|, |\Sigma|, |u|_{\mathrm{var}}, |h|]$-$Z$-$K$ is $W[1]$-hard.*

It shall be explained later in Section 3.2 that the nonerasing variants of the string morphism problem are trivially fixed parameter tractable if parameterised by $|w|$.

The reductions that have been used in order to prove Theorem 2 are of no use for proving Theorem 3, since they produce target strings whose lengths depend on the size of the graph and not on the size of the clique. For the proof of Theorem 3, we utilise the problem k-Multicoloured-Clique, i. e., we define a mapping Φ that maps a given graph $\mathcal{G} := (V, E)$ and a partition $V_1, V_2, \ldots, V_k$ of V to a source string $u \in (\Sigma \cup X)^+$ and a target string $w \in \Sigma^+$, where $\Sigma := \{\mathtt{a}_{\{i,j\}} \mid 1 \leq i \leq j \leq k, i \neq j\} \cup \{\$\}$ and $X := \{x_e \mid e \in E\}$. For the sake of concreteness, we define, for every i, $1 \leq i \leq k$, $V_i := \{v_{i,1}, v_{i,2}, \ldots, v_{i,t_i}\}$. Note that, for every i, j, $1 \leq i \leq j \leq k$, $i \neq j$, the symbols $\mathtt{a}_{\{i,j\}}$ and $\mathtt{a}_{\{j,i\}}$ are considered identical. For every i, j, $1 \leq i < j \leq k$, we define

$$\overline{u}_{i,j} := \$\, x_{e_{i,j,1}}\, x_{e_{i,j,2}} \ldots x_{e_{i,j,l_{i,j}}}\, \$ \text{ and } \overline{w}_{i,j} := \$\, \mathtt{a}_{\{i,j\}}\, \$\,,$$

where $e_{i,j,1}, e_{i,j,2}, \ldots, e_{i,j,l_{i,j}}$ is an enumeration of exactly the edges between V_i and V_j. Furthermore,

$$\overline{u} := \overline{u}_{1,2}\, \overline{u}_{1,3} \ldots \overline{u}_{1,k}\, \overline{u}_{2,3}\, \overline{u}_{2,4} \ldots \overline{u}_{2,k}\, \ldots\, \overline{u}_{k-1,k}\,,$$
$$\overline{w} := \overline{w}_{1,2}\, \overline{w}_{1,3} \ldots \overline{w}_{1,k}\, \overline{w}_{2,3}\, \overline{w}_{2,4} \ldots \overline{w}_{2,k}\, \ldots\, \overline{w}_{k-1,k}\,.$$

Next, for every i, $1 \leq i \leq k$, we define a gadget $(\widetilde{u}_i, \widetilde{w}_i)$ in the following way. For every j, p, $1 \leq p \leq t_i$, $1 \leq j \leq k$, $i \neq j$, we define $\widehat{u}_{i,p,j} := x_{e_{p,j,1}}\, x_{e_{p,j,2}} \ldots x_{e_{p,j,s_{p,j}}}$, where $e_{p,j,1}, e_{p,j,2}, \ldots, e_{p,j,s_{p,j}}$ is an enumeration of exactly the edges between vertex $v_{i,p}$ and some vertex of V_j. Next, for every p, $1 \leq p \leq t_i$, we define

$$\widehat{u}_{i,p} := \widehat{u}_{i,p,1}\, \widehat{u}_{i,p,2} \ldots \widehat{u}_{i,p,i-1}\, \widehat{u}_{i,p,i+1}\, \widehat{u}_{i,p,i+2} \ldots \widehat{u}_{i,p,k}\,.$$

We are now ready to define the gadget $(\widetilde{u}_i, \widetilde{w}_i)$:

$$\widetilde{u}_i := \widehat{u}_{i,1}^2\, \widehat{u}_{i,2}^2 \ldots \widehat{u}_{i,t_i}^2,$$
$$\widetilde{w}_i := (\mathtt{a}_{\{i,1\}}\, \mathtt{a}_{\{i,2\}} \ldots \mathtt{a}_{\{i,i-1\}}\, \mathtt{a}_{\{i,i+1\}}\, \mathtt{a}_{\{i,i+2\}} \ldots \mathtt{a}_{\{i,k\}})^2\,.$$

Furthermore, we define $\widetilde{u} := \$\, \widetilde{u}_1\, \$\, \widetilde{u}_2\, \$ \ldots \$\, \widetilde{u}_k\, \$$, $\widetilde{w} := \$\, \widetilde{w}_1\, \$\, \widetilde{w}_2\, \$ \ldots \$\, \widetilde{w}_k\, \$$ and, finally, $u := \overline{u}\, \widetilde{u}$ and $w := \overline{w}\, \widetilde{w}$.

▶ **Lemma 4.** *Let $\mathcal{G} := (V, E)$ be a graph, let $V_1, V_2, \ldots, V_k$ be a partition of V, such that every V_i is an independent set and let $(u, w) := \Phi(\mathcal{G}, V_1, V_2, \ldots, V_k)$. There exists a clique of size k in $\mathcal{G}$ if and only if there exists an* E*-injective substitution h of size* 1 *with $h(u) = w$.*

Proof Sketch. Every variable $x_e \in X$ corresponds to the edge $e \in E$. Due to the gadgets $(\overline{u}_{i,j}, \overline{w}_{i,j})$, $1 \leq i < j \leq k$, for every edge e between V_i and V_j the corresponding variable can either be substituted by symbol $\mathtt{a}_{\{i,j\}}$ or by the empty string ε, but, for every i, j, $1 \leq i \leq j \leq k$, $i \neq j$, exactly one variable corresponding to an edge between V_i and V_j must be mapped to $\mathtt{a}_{\{i,j\}}$. So mapping $\overline{u}$ to $\overline{w}$ corresponds to choosing exactly one edge between each two sets V_i and V_j, $1 \leq i \leq j \leq k$, $i \neq j$. In order to conclude that these edges are the edges of a clique, we somehow have to impose the condition that, for every i, $1 \leq i \leq k$, all the chosen edges connecting a vertex from V_i to vertices from each V_j, $1 \leq j \leq k$, $i \neq j$, are adjacent to exactly the same vertex of V_i. This condition is guaranteed by the gadgets $(\widetilde{u}_i, \widetilde{w}_i)$, $1 \leq i \leq k$, which can be seen in the following way. For a fixed i, $1 \leq i \leq k$, $\widetilde{u}_i$ contains all vertices between $v_{i,1}$ and vertices in V_1, then all vertices between $v_{i,1}$ and vertices in V_2 and so on until all vertices between $v_{i,1}$ and V_k are listed (this is

represented by $\widehat{u}_{i,1}$). Then, in the same way, all vertices between vertex $v_{i,2}$ and vertices in $V_1, V_2, \ldots, V_k$ are listed, and so on. The string $\widetilde{w}_i$, i. e., the counterpart to $\widetilde{u}_i$, contains a listing of edges between V_i and the other sets V_j, $1 \leq j \leq k$, $i \neq j$. The fact that all the $\widehat{u}_{i,p}$, $1 \leq p \leq t_i$, in $\widetilde{u}_i$ are squared and $\widetilde{w}_i$ is a square, too, allows us to conclude that the whole string $\widetilde{w}_i$ must be completely generated by one $\widehat{u}_{i,p}$, for some p, $1 \leq p \leq t_i$. This means that there is actually one distinct p, $1 \leq p \leq t_i$, such that *all* the edges between V_i and the other sets V_j, $1 \leq j \leq k$, $i \neq j$, are adjacent to this vertex $v_{i,p} \in V_i$. ◀

We note that Φ is an fpt-reduction with respect to the parameters $|w|$, $|\Sigma|$, $|u|_{\text{var}}$ and $|h|$.

▶ Proposition 5. Let $\mathcal{G}$ be a graph, let $V_1, V_2, \ldots, V_k$ be a partition of V and let $(u, w) := \Phi(\mathcal{G}, V_1, V_2, \ldots, V_k)$. Then $|w|$ and $|\Sigma|$ are bounded by a function of k and $|u|_{\text{var}} = 3$.

However, the reduction Φ only works for the problems Z-StrSubst, $Z \in \{\text{Inj}, \varepsilon\}$, but it can be extended to the problems Z-StrMorph, $Z \in \{\text{Inj}, \varepsilon\}$, as well, i. e., to a mapping Φ' that maps a k-Multicoloured-Clique instance to a source string $u' \in X^+$ and a target string $w' \in \Sigma^+$. To this end let $\mathcal{G} := (V, E)$ be a graph and let $V_1, V_2, \ldots, V_k$ be a partition of V, such that every V_i is an independent set. Since Φ' is very similar to Φ, we shall only point out in which regards they differ. The main difference is that for Φ', instead of using occurrences of the symbol \$ in u, we use an occurrence of a new variable per each occurrence of \$. Furthermore, in order to maintain the E-injectivity, each of these new variables has to match its own individual symbol in w. More formally, for every i, j, $1 \leq i < j \leq k$, we define $\overline{u}_{i,j} := z_{\mathfrak{c}_{i,j}}\, x_{e_{i,j,1}}\, x_{e_{i,j,2}} \ldots x_{e_{i,j,l_{i,j}}}\, z_{\mathfrak{c}_{i,j}}$, $\overline{w}_{i,j} := \mathfrak{c}_{i,j}\, \mathtt{a}_{\{i,j\}}\, \mathfrak{c}_{i,j}$ and $\widetilde{u} := z_{\$_1}\, \widetilde{u}_1\, z_{\$_2}\, \widetilde{u}_2\, z_{\$_3} \ldots z_{\$_k}\, \widetilde{u}_k\, z_{\$_{k+1}}$, $\widetilde{w} := \$_1\, \widetilde{w}_1\, \$_2\, \widetilde{w}_2\, \$_3 \ldots \$_k\, \widetilde{w}_k\, \$_{k+1}$, where the factors $\widetilde{u}_i$ and $\widetilde{w}_i$, $1 \leq i \leq k$, are defined as in the definition of Φ. Furthermore, analogously to the definition of Φ, we define $\overline{u}$ and $\overline{w}$ to be the concatenations of the factors $\overline{u}_{i,j}$ and $\overline{w}_{i,j}$, respectively. Finally, we define $u' := z_\% \, z_{\widehat{\%}}\, z_\% \, s \, z_{\widehat{\%}}\, \overline{u}\, z_{\widehat{\%}}\, \widetilde{u}$ and $w' := \% \, \widehat{\%} \, \% \, r \, \widehat{\%} \, \overline{u} \, \widehat{\%} \, \widetilde{w}$, where s is a concatenation of all the new variables of form $z_{\mathfrak{c}_{i,j}}$ and $z_{\$_i}$ and r is the corresponding concatenation of the symbols $\mathfrak{c}_{i,j}$ and $\$_i$. We note that Lemma 4 as well as Proposition 5 still hold with respect to Φ', which concludes the proof of Theorem 3.

▶ **Lemma 6.** *Let $\mathcal{G} := (V, E)$ be a graph, let $V_1, V_2, \ldots, V_k$ be a partition of V, such that every V_i is an independent set, let $(u', w') := \Phi'(\mathcal{G}, V_1, V_2, \ldots, V_k)$ and let $(u, w) := \Phi(\mathcal{G}, V_1, V_2, \ldots, V_k)$. There exists an* E*-injective morphism h with $h(u') = w'$ if and only if there exists an* E*-injective morphism g with $g(u) = w$.*

▶ Proposition 7. Let $\mathcal{G}$ be a graph, let $V_1, V_2, \ldots, V_k$ be a partition of V and let $(u', w') := \Phi'(\mathcal{G}, V_1, V_2, \ldots, V_k)$. Then $|w'|$ and $|\Sigma|$ are bounded by a function of k and $|u'|_{\text{var}} = 3$.

3.2 Fixed Parameter Tractability

We now present two brute-force algorithms for the string morphism problems. Let u be a source string and let w be a target string. The algorithm BF-1$_{\text{StrMorph}}$ on input (u, w) works as follows. All tuples $(w_1, w_2, \ldots, w_{|\text{var}(u)|})$ of factors of w are enumerated and if for such a tuple $h(u) = w$ holds, where $h(x_i) := w_i$, $1 \leq i \leq |\text{var}(u)|$, then the output is `YES` and `NO` otherwise. Obviously, BF-1$_{\text{StrMorph}}(u, w) =$ `YES` if and only if there exists a morphism h with $h(u) = w$. In an analogous way, for every $K \in$ SMP, we can define an algorithm BF-1$_K$, which solves the problem K. We only have to make sure that, depending on the problem K, we only enumerate m-tuples of factors of w that induce an injective, a nonerasing or an injective nonerasing morphism (or substitution).

▶ Proposition 8. Let $K \in$ SMP. The runtime of BF-1$_K(u, w)$ is $\mathrm{O}\left(|u| \times |w| \times \left(|w|^2\right)^{|\text{var}(u)|}\right)$.

We now slightly change the brute-force algorithm BF-1_K from above. To this end, let u be a source string, let w be a target string over some target alphabet Σ and let $k \in \mathbb{N}$. The algorithm BF-2_{StrMorph} on input (u, w, Σ, k) works just as BF-1_{StrMorph}, but instead of enumerating all tuples $(w_1, w_2, \ldots, w_{|\text{var}(u)|})$ of factors of w, it enumerates all tuples $(w_1, w_2, \ldots, w_{|\text{var}(u)|})$ of strings over Σ with $|w_i| \leq k$, $1 \leq i \leq |\text{var}(u)|$, and checks whether one of them induces a morphism h with $h(u) = w$. It can be easily verified that BF-$2_{\text{StrMorph}}(u, w, \Sigma, k) =$ `YES` if and only if there exists a morphism h of size k with $h(u) = w$. In a similar way as done for algorithm BF-1_{StrMorph}, for every $K \in$ SMP, we can extend BF-2_{StrMorph} to BF-2_K, which solves the problem K.

▶ Proposition 9. Let $K \in$ SMP. The runtime of the algorithm BF-$2_K(u, w, \Sigma, k)$ is $\mathrm{O}\left(|u| \times k \times \left(k \times |\Sigma|^k\right)^{|\text{var}(u)|}\right)$.

By applying the above brute-force algorithms for solving the string morphism problems, we can conclude the following fpt-results:

▶ **Theorem 10.**

A. *For every $K \in$ SMP, $[|\text{var}(u)|, |w|]$-K is in FPT.*
B. *For every $Z \in \{\text{Ne}, \text{Ne-Inj}\}$ and $K \in \{\text{StrMorph}, \text{StrSubst}\}$, $[|w|]$-Z-K is in FPT.*
C. *For every $K \in \{\text{StrMorph}, \text{StrSubst}\}$, $[|h|, |\Sigma|]$-Ne-Inj-K is in FPT.*
D. *For every $K \in$ SMP, $[|\text{var}(u)|, |h|]$-K is in FPT.*

Proof Sketch. Obviously, BF-1_K is an fpt-algorithm for $[|\text{var}(u)|, |w|]$-K, $K \in$ SMP, which proves A. For the Ne variants of the string morphism problems, we can assume $|w| \geq |u| \geq |\text{var}(u)|$; thus, for every $Z \in \{\text{Ne}, \text{Ne-Inj}\}$ and $K \in \{\text{StrMorph}, \text{StrSubst}\}$, BF-$1_{Z\text{-}K}(u, w)$ has a runtime of $\mathrm{O}(|u| \times |w| \times (|w|^2)^{|w|}))$, which proves B.

For every $k \in \mathbb{N}$, there are $l := \sum_{i=1}^{k} |\Sigma|^i$ different non-empty strings over Σ with length at most k. Thus, if $|\text{var}(u)| > l$, then every morphism h with $|h| \leq k$ is necessarily non-injective and if, on the other hand, $|\text{var}(u)| < l$, then BF-$2_{\text{Ne-Inj-}K}(u, w, \Sigma, k)$, $K \in \{\text{StrMorph}, \text{StrSubst}\}$, has an fpt-runtime since $|\text{var}(u)| \leq l$. This proves C.

Finally, in order to prove D, we note that, for every $k \in \mathbb{N}$, a morphism or a substitution h of size k can introduce at most $|\text{var}(u)| \times k$ new symbols. Hence, for every $K \in$ SMP, BF-$2_K(u, w, \Gamma, k)$ solves $[|\text{var}(u)|, |h|]$-K in fpt-time, where Γ (with $|\Gamma| \leq |\text{var}(u)| \times k$) is the alphabet over which the images with respect to h are defined. ◀

We conclude this section by pointing out that the results presented in Section 3.1 and 3.2 completely settle the fixed parameter tractability of all possible parameterised variants of string morphism problems, with respect to the parameters considered in the context of this work. In order to verify this claim, it is helpful to recall these results in form of a table.

Problems	$\lvert\text{var}(u)\rvert$	$\lvert w\rvert$	$\lvert u\rvert_{\text{var}}$	$\lvert h\rvert$	$\lvert\Sigma\rvert$	Complexity	Reference
SMP	p	p	–	–	–	FPT	Thm. 10.A
{Ne, Ne-Inj}-{StrMorph, StrSubst}	–	p	–	–	–	FPT	Thm. 10.B
Ne-Inj-{StrMorph, StrSubst}	–	–	–	p	p	FPT	Thm. 10.C
SMP	p	–	–	p	–	FPT	Thm. 10.D
SMP	p	–	p	–	6	$W[1]$-hard	Thm. 2
$\{\varepsilon$, Inj$\}$-{StrMorph, StrSubst}	–	p	3	1	p	$W[1]$-hard	Thm. 3

In the above table, an entry p means that the problems denoted in the row are parameterised by the parameter in the column and an integer entry constitutes a constant bound for this parameter. We note that all the cases parameterised by both $|\text{var}(u)|$ and $|w|$ are settled

by row 1. Furthermore, all the cases parameterised by $|w|$, but not by $|\mathrm{var}(u)|$ are settled by rows 2 and 6, and all the cases parameterised by $|\mathrm{var}(u)|$, but not by $|w|$ are settled by rows 4 and 5. In order to see that all the cases parameterised neither by $|\mathrm{var}(u)|$ nor by $|w|$ are settled as well, we need to take a closer look.

From row 5, we can only conclude that as long as $|h|$ is not a parameter, then all variants are $W[1]$-hard. However, for the cases where $|h|$ is a parameter, we can only conclude from row 6 the $W[1]$-hardness for all but the NE and NE-INJ variants, and, in addition to that, from row 3 we can conclude the FPT-membership for the NE-INJ variant, where $|\Sigma|$ is a parameter, too. Consequently, for every $K \in \{\text{STRMORPH}, \text{STRSUBST}\}$ and $Z \in \{\text{NE}, \text{NE-INJ}\}$, the following cases are open: (1) $[|h|, |u|_{\mathrm{var}}, |\Sigma|]$-NE-$K$, (2) $[|h|, |\Sigma|]$-NE-K, (3) $[|h|, |u|_{\mathrm{var}}]$-$Z$-$K$ and (4) $[|h|]$-Z-K. In [13] it has been shown that the problems NE-K are $\mathcal{NP}$-complete even if the parameters $|h|$, $|u|_{\mathrm{var}}$ and $|\Sigma|$ are bounded by constants, which implies that, unless $\mathcal{P} = \mathcal{NP}$, the problems of cases (1) and (2) are not in XP; thus, they are not in FPT. The same holds for the problems Z-K with respect to parameters $|h|$ and $|u|_{\mathrm{var}}$, which, in a similar way, implies that the problems of cases (3) and (4) are not in FPT.

3.3 $W[1]$-Membership and $W[P]$-Membership

In this section, we investigate the $W[1]$-membership and $W[P]$-membership for the $W[1]$-hard variants of string morphism problems. In this regard, we first explain why the problems $[|\mathrm{var}(u)|, |u|_{\mathrm{var}}]$-$K$ and $[|w|]$-K, $K \in \mathrm{SMP}$, are in $W[1]$.

▶ **Theorem 11.** *Let $K \in \mathrm{SMP}$. The problems $[|\mathrm{var}(u)|, |u|_{\mathrm{var}}]$-$K$ and $[|w|]$-K are in $W[1]$.*

Proof Sketch. We sketch a reduction from the problem $[|\mathrm{var}(u)|, |u|_{\mathrm{var}}]$-STRSUBST to the problem SHORT-NTM-COMP, which can be extended to the other problems $[|\mathrm{var}(u)|, |u|_{\mathrm{var}}]$-$K$, $K \in \mathrm{SMP}$. To this end, let $u := u_0\, y_1\, u_1\, y_2\, u_2\, y_3 \ldots y_n\, u_n$, $y_i \in X$, $1 \leq i \leq n$, $u_j \in \Sigma^*$, $0 \leq j \leq n$, be the source string and let $w \in \Sigma^*$ be the target string. The Turing machine starts with the string $u' := y_1 y_2 \ldots y_n$ on the tape. It then guesses a subset $S \subseteq \mathrm{var}(u)$ and replaces every occurrence of a $y_i \in S$ by T_ε (which means that this variable is erased). Every occurrence of a variable $y_i \notin S$ is replaced by $T^{(i)}_{j,k}$, for some j, k, $1 \leq j \leq k \leq |w|$ (which means that the i^{th} occurrence of a variable is allocated to factor $w[j,k]$ of the target string). It now only remains to check, for every factor $T^{(i_1)}_{j_1,k_1}\, T^{l}_{\varepsilon}\, T^{(i_2)}_{j_2,k_2}$, $0 \leq l \leq n-2$, whether $u_{i_1}\, u_{i_1+1} \ldots u_{i_2-1} = w[k_1+1, j_2-1]$, and to check, for every i_1, i_2, $1 \leq i_1 < i_2 \leq n$, with $u'[i_1] = T^{(i_1)}_{j_1,k_1}$, $u'[i_2] = T^{(i_2)}_{j_2,k_2}$ and $y_{i_1} = y_{i_2}$, whether $w[j_1,k_1] = w[j_2,k_2]$. All the data that the Turing machine requires to perform these checking procedures can be computed beforehand in polynomial time by the transformation machine; thus, the number of steps of the Turing machine is bounded by $g(|u'|)$, for some function g. Since $|u'| \leq |u|_{\mathrm{var}} \times |\mathrm{var}(u)|$, the number of steps is bounded by a function of the parameters. This reduction can be easily adapted to all the other problems $[|\mathrm{var}(u)|, |u|_{\mathrm{var}}]$-$K$, $K \in \mathrm{SMP}$.

In order to reduce the problems $[|w|]$-STRSUBST to SHORT-NTM-COMP, we need a slightly different approach: the Turing machine performs its computation on the target string instead on the source string. More precisely, the Turing machine starts with the target string w on the tape, nondeterministically initialises a counter c, $1 \leq c \leq |u|$, and then moves over w from left to right. In every step, a terminal symbol is replaced by the number c and then c is nondeterministically set to a value j, $c \leq j \leq |u|$. After this procedure, the input tape contains an increasing sequence of $|w|$ numbers between 1 and $|u|$. Every consecutive sequence of occurrences of the same number i, $1 \leq i \leq |u|$, means that the corresponding factor of w is "produced" by $u[i]$ (note that this can be a variable or a terminal symbol). Now it only needs to be checked whether this allocation of factors from w to the symbols of

u induces a substitution that maps u to w. All the data required for this checking procedure can be computed beforehand in polynomial time by the transformation machine; thus, the number of steps of the Turing machine is bounded by $g(|w|)$, for some function g. It is straightforward to adapt this reduction to the other problems $[|\mathrm{var}(w)|]$-K, $K \in \mathrm{SMP}$. ◀

We wish to point out that if a problem L-K, $K \in \mathrm{SMP}$, has shown to be in $W[1]$, then this $W[1]$-membership is preserved if we add other parameters to L. Hence, Theorem 11 implies $W[1]$-completeness for all the $W[1]$-hard versions of string morphism problems, except $[|\mathrm{var}(u)|, |\Sigma|]$-$K$ and $[|\mathrm{var}(u)|]$-K, $K \in \mathrm{SMP}$, for which $W[1]$-membership does not follow, since in the above reduction, we need $|u|_{\mathrm{var}}$ as a parameter, too. However, for the problems $[|\mathrm{var}(u)|]$-K, $K \in \mathrm{SMP}$, we can show a weaker result, i. e., their $W[P]$-membership (which then also carries over to the problems $[|\mathrm{var}(u)|, |\Sigma|]$-$K$, $K \in \mathrm{SMP}$).

▶ **Theorem 12.** *For every* $K \in \mathrm{SMP}$, $[|\mathrm{var}(u)|]$-$K \in W[P]$.

Proof Sketch. A problem Π parameterised by k is in $W[P]$ if and only if there is a computable function h, a polynomial p and a nondeterministic Turing machine M deciding Π such that on every run with input w the machine M performs at most $p(|w|)$ steps, at most $h(k) \times \log(|w|)$ of them being nondeterministic (see Flum and Grohe [14]).

We only show how $[|\mathrm{var}(u)|]$-Ne-StrMorph can be solved by a Turing machine with the above properties (this procedure can be easily adapted to all other cases). Let $u \# w$ be the input, where $u = y_1\, y_2 \ldots y_m$, $y_i \in \mathrm{var}(u)$, $1 \leq i \leq m$, is the source string and w is the target string. The Turing machine nondeterministically guesses numbers $l_1, l_2, \ldots, l_{|\mathrm{var}(u)|} \in \mathbb{N}$, which induce a factorisation of w that fits to u, i. e., $w = w_1 w_2 \ldots w_m$, where, for every i, $1 \leq i \leq m$, j, $1 \leq j \leq |\mathrm{var}(u)|$, $|w_i| = l_j$ if $y_i = x_j$. The Turing machine then accepts if, for every p, q, $1 \leq p < q \leq |u|$, $y_p = y_q$ implies $w_p = w_q$, and rejects otherwise. The correctness of this procedure is obvious, it only remains to show that it satisfies the above mentioned conditions. All the tasks that need to be performed by the Turing machine can be carried out in time polynomial in $|w|$ and $|u|$. Furthermore, the only nondeterministic steps are the ones that are performed in order to guess the factorisation of w and these are $\mathrm{O}(|\mathrm{var}(u)| \times \log(|w|))$ many, since the factorisation is completely determined by the lengths of the $|\mathrm{var}(u)|$ factors and each of these lengths is bounded by $|w|$. ◀

4 A Lower Bound

For most string problems, it is a natural assumption that the alphabet Σ is fixed (in fact, it often has very small cardinality as, e. g., 2 if we are dealing with binary numbers or 4 in the case of DNA sequences). Furthermore, if we use strings with variables (i. e., source strings) for specifying a class of similar string objects (which is a typical application of strings with variables), then, for many applications, there are only finitely many string objects that can replace the variables. Hence, the problems $[|h| \leq k_1, |\Sigma| \leq k_2]$-$K$, $K \in \mathrm{SMP}$, $k_1, k_2 \in \mathbb{N}$ (i. e., the parameters $|h|$ and $|\Sigma|$ are bounded by k_1 and k_2, respectively), are of special interest. We recall that Proposition 9 demonstrates that, for every constants $k_1, k_2 \in \mathbb{N}$ and for every $K \in \mathrm{SMP}$, $[|h| \leq k_1, |\Sigma| \leq k_2]$-$K$ can be solved in time $\mathrm{O}(|u| \times k_1 \times (k_1 \times k_2^{k_1})^{|\mathrm{var}(u)|}) = |u| \times 2^{\mathrm{O}(|\mathrm{var}(u)|)}$. Next, we show that if $k_2 \geq 2$, then, for every $K \in \{\mathrm{StrMorph}, \mathrm{StrSubst}\}$, there does not exist an algorithm that solves $[|h| \leq k_1, |\Sigma| \leq k_2]$-$K$ in time $(|u||w|)^{\mathrm{O}(1)} \times 2^{\mathrm{o}(|\mathrm{var}(u)|)}$, unless ETH fails.

To this end, we define a reduction Φ from 3SAT to StrSubst. Let $C := \{c_1, c_2, \ldots, c_m\}$ be a set of three-literal-clauses with variables $v_1, v_2, \ldots, v_n$ ($\neg v_i$ denotes the negation of v_i). We define $\overline{u} := \mathtt{¢}\, x_1\, \overline{x}_1\, \mathtt{¢}\, x_2\, \overline{x}_2\, \mathtt{¢} \ldots \mathtt{¢}\, x_n\, \overline{x}_n\, \mathtt{¢}$, $\overline{w} := \mathtt{¢}\, (\mathtt{a}\, \mathtt{¢})^n$ and, for every clause $c_i :=$

$\{p_{i_1}, p_{i_2}, p_{i_3}\}$, $1 \leq i \leq m$, we define $\widehat{u}_i := \mathtt{c}\, y_{i_1}\, y_{i_2}\, y_{i_3}\, z_{i,1}\, z_{i,2}\, \mathtt{c}\, z_{i,1}\, z_{i,2}\, z'_{i,1}\, z'_{i,2}\, \mathtt{c}$ and $\widehat{w}_i := \mathtt{c}\,\mathtt{a}\,\mathtt{a}\,\mathtt{a}\,\mathtt{c}\,\mathtt{a}\,\mathtt{a}\,\mathtt{c}$, where $y_{i_j} := x_{i_j}$ if $p_{i_j} = v_{i_j}$ and $y_{i_j} := \overline{x}_{i_j}$ if $p_{i_j} = \neg v_{i_j}$, $1 \leq j \leq 3$. Finally, $u := \overline{u}\,\widehat{u}_1\,\widehat{u}_2 \ldots \widehat{u}_m$, $w := \overline{w}\,\widehat{w}_1\,\widehat{w}_2 \ldots \widehat{w}_m$ are the source and target strings constructed by Φ.

▶ **Lemma 13.** *Let C be a 3CNF formula and let $(u, w) := \Phi(C)$. The formula C is satisfiable if and only if there exists a substitution h of size 1 with $h(u) = w$.*

We note that $\Phi(C)$ produces a source string u with $|u| = 3n + 1 + 12m$ and a target string $w \in \{\mathtt{a}, \mathtt{c}\}^*$ with $|w| = 2n + 1 + 8m$, where n is the number of Boolean variables and m is the number of clauses of C. This implies that, for every $k_1, k_2 \in \mathbb{N}$, $k_2 \geq 2$, if $[|h| \leq k_1, |\Sigma| \leq k_2]$-StrSubst can be solved in time $(|u||w|)^{\mathrm{O}(1)} \times 2^{\mathrm{o}(|\mathrm{var}(u)|)}$, then 3SAT can be solved in time $(m+n)^{\mathrm{O}(1)} \times 2^{\mathrm{o}(m+n)}$.

Furthermore, the reduction Φ can be extended to morphisms, i. e., to a reduction Φ' that maps a Boolean formula C with n variables and m clauses to a source string u that only contains variables, i. e., $u \in X^*$, with $|u| = \mathrm{O}(n) + \mathrm{O}(m)$. To this end, let C be a set of m three-literal-clauses with n variables and let $(u, w) := \Phi(C)$. First, we obtain a source string $u' \in X^*$ from u by substituting every occurrence of $\mathtt{c}$ by an occurrence of the new variable $y_{\mathtt{c}}$. Next, we define $u'' := y_{\mathtt{c}}\, y_{\mathtt{c}}\, (u')^2$ and $w'' := \mathtt{c}\,\mathtt{c}\,(w)^2$. Obviously, $|u''| = 2|u| + 2 = \mathrm{O}(n) + \mathrm{O}(m)$. In order to prove that Φ' is a valid reduction, it is sufficient to show that there exists a substitution h of size 1 with $h(u) = w$ if and only if there exists a morphism g of size 1 with $g(u'') = w''$. The *only if* direction is obvious, since if $h(u) = w$, then $g(u'') = w''$, where $g(x) := h(x)$, $x \in \mathrm{var}(u)$, and $g(y_{\mathtt{c}}) := \mathtt{c}$. For the *if* direction, we observe that if $g(u'') = w''$ and $g(y_{\mathtt{c}}) = \mathtt{c}$, then $g(u) = w$ holds as well. If, on the other hand, $g(y_{\mathtt{c}}) = \varepsilon$, then $g(u^2) = w''$, which is a contradiction, since w'' is not a square.

Hence, for every $k_1, k_2 \in \mathbb{N}$, $k_2 \geq 2$, if we can solve $[|h| \leq k_1, |\Sigma| \leq k_2]$-$K$, $K \in \{\text{StrMorph}, \text{StrSubst}\}$, in time $(|u||w|)^{\mathrm{O}(1)} \times 2^{\mathrm{o}(|\mathrm{var}(u)|)}$, then 3SAT can be solved in time $(m+n)^{\mathrm{O}(1)} \times 2^{\mathrm{o}(m+n)}$. Furthermore, by applying the *Sparsification Lemma* (see [21]), we can obtain the following result.

▶ **Theorem 14.** *For every $K \in \{\text{StrMorph}, \text{StrSubst}\}$ and $k_1, k_2 \in \mathbb{N}$, $k_2 \geq 2$, $[|h| \leq k_1, |\Sigma| \leq k_2]$-$K$ cannot be solved in time $(|u||w|)^{\mathrm{O}(1)} \times 2^{\mathrm{o}(|\mathrm{var}(u)|)}$, unless ETH fails.*

References

1. F. N. Abu-Khzam, H. Fernau, M. A. Langston, S. Lee-Cultura, and U. Stege. A fixed-parameter algorithm for string-to-string correction. *Discrete Optimization*, 8:41–49, 2011.
2. A. Amir and I. Nor. Generalized function matching. *Journal of Discrete Algorithms*, 5:514–523, 2007.
3. D. Angluin. Finding patterns common to a set of strings. In *Proc. 11th Annual ACM Symposium on Theory of Computing, STOC 1979*, pages 130–141, 1979.
4. B. S. Baker. Parameterized pattern matching: Algorithms and applications. *Journal of Computer and System Sciences*, 52:28–42, 1996.
5. L. Cai, J. Chen, R. G. Downey, and M. R. Fellows. On the parameterized complexity of short computation and factorization. *Archive for Mathematical Logic*, 36:321–337, 1997.
6. C. Câmpeanu, K. Salomaa, and S. Yu. A formal study of practical regular expressions. *International Journal of Foundations of Computer Science*, 14:1007–1018, 2003.
7. M. Cesati. The Turing way to parameterized complexity. *Journal of Computer and System Sciences*, 67:654–685, 2003.
8. R. Clifford, A. W. Harrow, A. Popa, and B. Sach. Generalised matching. In *Proc. 16th International Symposium on String Processing and Information Retrieval, SPIRE 2009*, volume 5721 of *LNCS*, pages 295–301, 2009.

9 R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.

10 R.G. Downey, M.R. Fellows, B. Kapron, M.T. Hallett, and H.T. Wareham. Parameterized complexity of some problems in logic and linguistics (extended abstract). In *Proc. 2nd Workshop on Structural Complexity and Recursion-theoretic Methods in Logic Programming*, volume 813 of *LNCS*, pages 89–101, 1994.

11 A. Ehrenfeucht and G. Rozenberg. Finding a homomorphism between two words is NP-complete. *Information Processing Letters*, 9:86–88, 1979.

12 M. R. Fellows, D. Hermelin, F. Rosamond, and S. Vialette. On the parameterized complexity of multiple-interval graph problems. *Theoretical Computer Science*, 401:53–61, 2009.

13 H. Fernau and M. L. Schmid. Pattern matching with variables: A multivariate complexity analysis. In *Proc. 24th Annual Symposium on Combinatorial Pattern Matching, CPM 2013*, volume 7922 of *LNCS*, pages 83–94, 2013.

14 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

15 F. V. Fomin and D. Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. Springer, 2010.

16 D. D. Freydenberger, D. Reidenbach, and J. C. Schneider. Unambiguous morphic images of strings. *International Journal of Foundations of Computer Science*, 17:601–628, 2006.

17 M. R. Garey and D. S. Johnson. *Computers And Intractability*. W. H. Freeman and Company, 1979.

18 M. Geilke and S. Zilles. Learning relational patterns. In *Proc. 22nd International Conference on Algorithmic Learning Theory, ALT 2011*, volume 6925 of *LNCS*, pages 84–98, 2011.

19 T. Harju and J. Karhumäki. Morphisms. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, chapter 7, pages 439–510. Springer, 1997.

20 O. Ibarra, T.-C. Pong, and S. Sohn. A note on parsing pattern languages. *Pattern Recognition Letters*, 16:179–182, 1995.

21 R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63:512–530, 2001.

22 T. Jiang, E. Kinber, A. Salomaa, K. Salomaa, and S. Yu. Pattern languages with and without erasing. *International Journal of Computer Mathematics*, 50:147–163, 1994.

23 D. Lokshtanov, D. Marx, and S. Saurabh. Lower bounds based on the Exponential Time Hypothesis. *EATCS Bulletin*, 105:41–72, 2011.

24 A. Mateescu and A. Salomaa. Finite degrees of ambiguity in pattern languages. *RAIRO Informatique théoretique et Applications*, 28:233–253, 1994.

25 D. Reidenbach and M. L. Schmid. A polynomial time match test for large classes of extended regular expressions. In *Proc. 15th International Conference on Implementation and Application of Automata, CIAA 2010*, volume 6482 of *LNCS*, pages 241–250, 2011.

26 D. Reidenbach and M. L. Schmid. Patterns with bounded treewidth. In *Proc. 6th International Conference on Language and Automata Theory and Applications, LATA 2012*, volume 7183 of *LNCS*, pages 468–479, 2012.

27 M. L. Schmid. *On the Membership Problem for Pattern Languages and Related Topics*. PhD thesis, Dept. of Computer Science, Loughborough University, 2012.

28 T. Shinohara. Polynomial time inference of pattern languages and its application. In *Proc. 7th IBM Symposium on Mathematical Foundations of Computer Science*, pages 191–209, 1982.

29 F. Stephan, R. Yoshinaka, and T. Zeugmann. On the parameterised complexity of learning patterns. In *Proc. 26th International Symposium on Computer and Information Sciences, ISCIS 2011*, pages 277–281.